Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

# по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний в графе»

**Выполнили:**

студенты группы 20ВВ3

Халилов Алмаз

Китаев Андрей

Новиков Иван

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пеза 2021

# **Название**

Поиск расстояний в графе

**Цель работы** – изучение поиска расстояний в графе. Осуществление процедуры поиска расстояний.

# **Методические указания**

Поиск расстояний – довольно распространенная задача анализа графов.

Для поиска расстояний можно использовать процедуры обхода графа. Для этого при каждом переходе в новую вершину необходимо запоминать, сколько шагов до нее мы сделали. При этом вектор, который хранил информацию о посещении вершин становится вектором расстояний. Довольно просто модернизировать для поиска расстояний в графе алгоритм обхода в ширину, т.к. этот алгоритм проходит вершины по уровням удаленности, то для не ориентированного графа для вершин каждого следующего уровня глубины расстояние от исходной вершины увеличивается на 1. Удалённость в данном случае понимается как количество ребер, по которым необходимо прейти до достижения вершины.

Таким образом, можно предложить следующую реализацию алгоритма обхода в ширину.

**Вход**: G – матрица смежности графа, v – исходная вершина.

**Выход**: DIST – вектор расстояний до всех вершин от исходной.

**Алгоритм ПОШ**

1.1. для всех i положим DIST [i] = -1 пометим как "не посещенную";

1.2. **ВЫПОЛНЯТЬ** BFSD (v).

1.3 для всех i вывести DIST [i] на экран;

**Алгоритм** BFSD(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

2.3. Обновить вектор расстояний DIST [ x ] = 0;

2.4. **ПОКА**  Q != ∅ очередь не пуста **ВЫПОЛНЯТЬ**

2.5. v = Q.front() установить текущую вершину;

2.6. Удалить первый элемент из очереди Q.pop();

2.7. вывести на экран v;

2.8. **ДЛЯ** i = 1 **ДО** size\_G **ВЫПОЛНЯТЬ**

2.9. **ЕСЛИ** G(v,i) = = 1**И** DIST = = -1

2.10. **ТО**

2.11. Поместить i в очередь Q.push(i);

2.12. Обновить вектор расстояний DIST [ i ] = DIST [ v ] + 1;

Реализация состоит из подготовительной части, в которой все вершины помечаются как не поcещенные (п.1.1). В отличие от алгоритма BFS не посещенные вершины помечаем -1, т.к. значение 0 и 1 могут быть расстояниями. Расстояние 0 – от исходной вершины до самой себя.

В самой процедуре как и в алгоритме BFS сначала создается пустая очередь (п. 2.1), в которую помещается исходная вершина, из которой начат обход (п.2.2). Расстояние до этой вершины (п.2.3) устанавливается равным 0 (расстояние до самой себя).

Далее итерационно, пока очередь не опустеет, из нее извлекается первый элемент, который становится текущей вершиной (п. 2.5, 2.6). Затем в цикле просматривается **v**-я строка матрицы смежности графа G(v,i). Как только алгоритм встречает смежную с **v** не посещенную вершину (п.2.9), эта вершина помещается в очередь (п.2.11) и для нее обновляется вектор расстояния (п.2.12). Расстояние до новой **i**-й вершины вычисляется как расстояние до текущей **v**-й вершины плюс 1 (так как ребра нашего графа не взвешенные).

После просмотра строки матрицы смежности алгоритм делает следующую итерацию цикла 2.4 или заканчивает работу, если очередь пуста.

Таким образом, если вершина помещается в очередь при просмотре сроки матрицы смежности на 1-й итерации, то они находятся на 1 уровне удаленности и расстояние до этих вершин будет равным 1.

DIST [ i ] = DIST [ v ] + 1, где DIST [ v ] = 0 – расстояние от исходной вершины до самой себя.

Далее, начинают просматриваться вершины первого уровня и соответствующие им строки матрицы смежности. При добавлении смежных с вершинами первого уровня вершин, расстояния до них будут равны 2.

DIST [ i ] = DIST [ v ] + 1, где DIST [ v ] = 1 – расстояние от исходной вершины до вершин 1 уровня.

После того, как все вершины первого уровня будут просмотрены и извлечены из очереди, начнется просмотр вершин 2 уровня и соответствующих им строк матрицы смежности. При добавлении смежных с вершинами второго уровня вершин, расстояния до них будут равны 3.

DIST [ i ] = DIST [ v ] + 1, где DIST [ v ] = 2 – расстояние от исходной вершины до вершин 2 уровня.

И так далее, алгоритм проходит вершины по уровням, пока очередь не опустеет.

**Лабораторное задание**

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

3.\* Реализуйте процедуру поиска расстояний для графа, представленного списками смежности.

**Ход Работы**

**Листинг**

#define \_CRT\_NONSTDC\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#include <string.h>

#include <windows.h>

#include <queue>

using namespace std;

void matrix(int\*\* g, int size)

{

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

g[i][j] = rand() % 2;

}

}

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

if (i == j)

g[i][j] = 0;

else

g[i][j] = g[j][i];

}

}

printf("Матрица: \n");

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

printf("%2d", g[i][j]);

}

printf("\n");

}

}

void F(int\*\* g, int\* NUM, int v, int size)

{

queue <int> q;

q.push(v);

NUM[v] = 0;

while (!q.empty())

{

v = q.front();

q.pop();

//printf("%d ", NUM[v]);

for (int i = 0; i < size; i++)

{

if ((g[v][i] == 1) && (NUM[i] == -1))

{

q.push(i);

NUM[i] = 1 + NUM[v];

}

}

}

printf("\n\n");

for (int i = 0; i < size; i++)

{

printf("%d ", NUM[i]);

}

}

void main()

{

setlocale(LC\_ALL, "RUS");

int i = 0;

int j = 0;

int size = 0;

int\*\* g = 0;

int v = 0;

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

printf("Введите размер матрицы: ");

scanf("%d", &size);

g = (int\*\*)malloc(size \* sizeof(int\*));

for (i = 0; i < size; i++)

{

g[i] = (int\*)malloc(size \* sizeof(int));

}

matrix(g, size);

printf("Точка входа: ");

scanf("%d", &v);

int\* NUM = (int\*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

{

NUM[i] = -1;

}

F(g, NUM, v, size);

printf("\n");

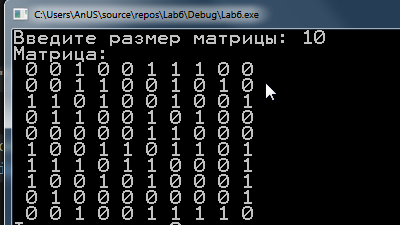
system("pause");

}

**Задание 1:**

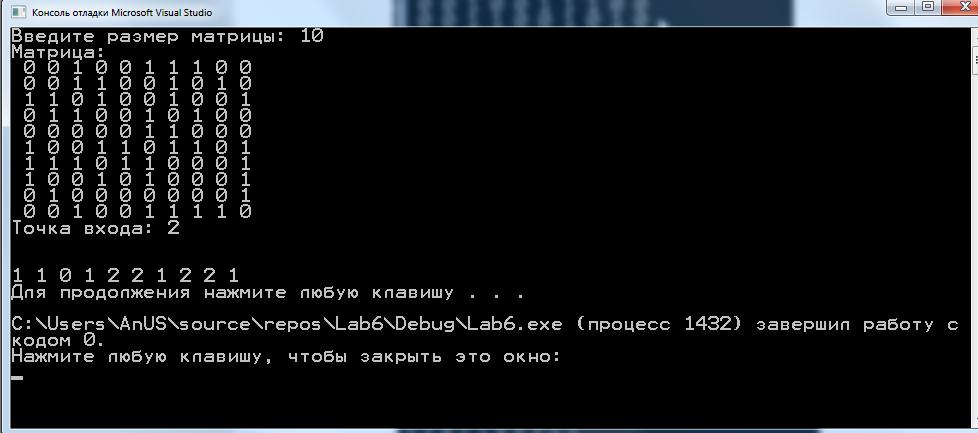
1. Сгенерировали (используя генератор случайных чисел) матрицу

смежности для неориентированного графа G и вывели матрицу на экран.



*Рисунок 1. Генерация массива*

2. Для сгенерированного графа осуществили процедуру поиска расстояний используя класс **queue** из стандартной библиотеки С++.



*Рисунок 2. Ввод точки входа и результат выполнения программы.*

**Вывод:** изучили поиск расстояний в графе – одну из наиболее распространенных операций с графами. Написали и протестировали программу, которая создает граф определенного размера и ищет расстояние.